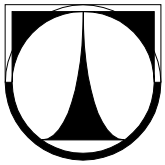
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií



BAKALÁŘSKÁ PRÁCE

Liberec 2008 **Josef Minařík**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektrotechnika a informatika
Studijní obor: 18024022 – Informatika a logistika

Shlukování výsledků vyhledávání

Clustering of search results

Bakalářská práce

Autor: **Josef Minařík**
Vedoucí práce: Mgr. Jiří Vraný
Konzultant: Ing. Michal Bukovský

V Liberci 16. 5. 2008

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum: 16. 5. 2008

Podpis:

Poděkování

Rád bych vyjádřil své poděkování Ing. Michalu Bukovskému za odborné konzultace a množství poskytnuté literatury, ze které jsem mohl čerpat. Dále bych chtěl poděkovat společnosti Seznam, a. s. za data poskytnutá pro účely testování.

Abstrakt

Automatická klasifikace dat bez znalosti vnitřní struktury, především pak dat textových, prochází neustálým vývojem, jelikož se stále objevují nové a nové výzvy, se kterým je potřeba se vypořádat. Tato práce se snaží snaží zmapovat problematiku vyhledávání informací a vybrat vhodné algoritmy pro shlukování výsledků vyhledávání. Shlukování nikdy nestojí v dokumentografických systémech samostatně, vždy je velmi těsně napojeno na ostatní procesy systému. Proto se práce snaží odkrýt pozadí takových procesů a jejich vzájemné vztahy. Začíná problematikou vyhledávání, mapuje datové struktury a informuje o výzvách na poli vyhledávacích strojů. Další kapitola rozebírá shlukování obecně, představuje základní algoritmy rozdělené do kategorií od tradičních po nově vzniklé a speciální význam klade právě tématu shlukování výsledků vyhledávání. Ve druhé části práce informuje o implementaci vybraných shlukovacích algoritmů a výsledcích jejich testování. Popisuje hlavní metody vytvořených aplikací. Pokračuje způsobem provádění testů a vizualizací výstupů z programů. Vyhodnocení experimentů se shlukovacími algoritmy nabízí relativně překvapivé výsledky.

Abstract

Automatic classification of data, especially textual data, without knowing its inner structure evolves continually because there are new and new challenges appearing that are necessary to deal with. This work tries to map information retrieval problems and to choose suitable algorithms for clustering of search results. Clustering never stands separately in documentographic systems as it is always in a tight cooperation with other processes in the system. That is why this work tries to disclose the background of those processes and their relationships. It starts with data retrieval problems, maps data structures and informs about challenges related to search engines. Next chapter analyze clustering in general, it presents basic algorithms divided into categories, it goes from traditional to recently developed and it puts the special accent on the clustering of search results topic. The second part of the work informs about an implementation of selected clustering algorithms and about its testing results. It describes main methods of the developed applications. Then it explains the way of testing and visualisation output of the programs. Evaluation of experiments with clustering algorithms shows relatively surprising results.

Obsah

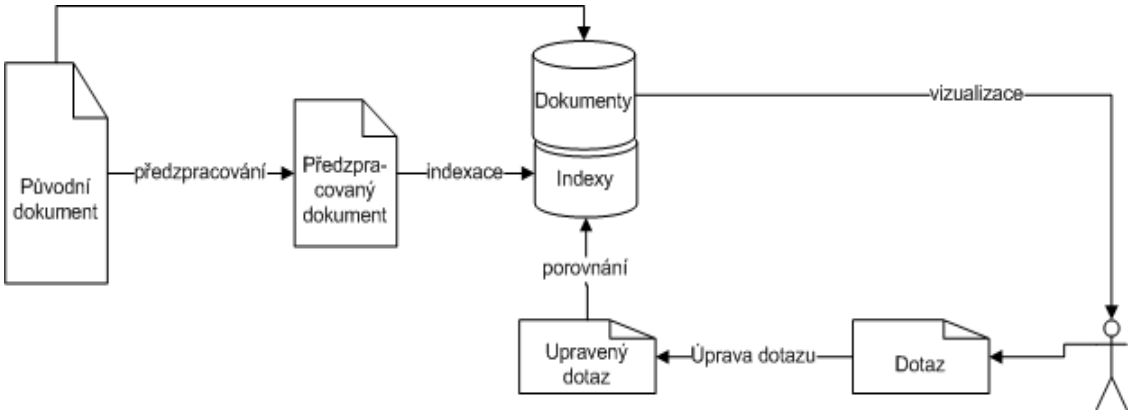
1 Úvod

Je tomu již drahně let, co internet opustil výhradně akademickou půdu. Od té doby jeho popularita neustále roste, což mimo jiné dokazují čísla ze serveru internetworldstats.com, podle nějž se celosvětově zvýšil počet lidí využívajících internet mezi roky 2000 a 2008 o 265,6 %. Internet se stal za dobu své existence díky World Wide Web (WWW) takřka bezednou truhlicí veřejně přístupných informací. Počet lidí připojených do sítě se musí logicky promítnout do objemu dat dostupných přes WWW. Na celkovém množství dat se velkou měrou podílejí multimediální soubory (obrázky, zvuk, video) a další binární data, nicméně klíčovou roli hrají data reprezentovaná prostým textem, ať už se jedná o reklamní texty, komentáře, odborné statě, encyklopedické texty, novinové články a podobně. Text je obecně dobře čitelný jak počítačem, tak člověkem. Toho se s výhodou využívá při vyhledávání informací na webu. Nicméně vzhledem k množství dat se vyhledávací stroje musí vypořádat s otázkami relevance a zobrazení výsledků vyhledávání a s rychlostí procházení obsáhlých databází. V tomto směru již byla provedena řada výzkumů zaměřených na využití shlukování pro optimalizaci (slučování stejných nebo podobných dokumentů do skupin za účelem snížení počtu porovnávání) a vizualizaci (výpis nalezených dokumentů bez duplicit nebo rozdělení dokumentů do skupin podle témat) výsledků vyhledávání.

2 Vyhledávání

Vyhledávání informací, anglicky information retrieval (IR), je na poli informačních technologií stále obrovskou výzvou, která trvá již od 50. let 20. století jako důsledek vzniku a vývoje dokumentografických (textových) informačních systémů (DIS), tedy původně nástrojů automatizace postupů používaných v knihovnictví. Takovéto systémy slouží ke zpracování textových informací v přirozeném jazyce bez pevné vnitřní struktury. Stejně tak se tato kapitola zabývá vyhledáváním informací pouze v textové podobě.

Celý DIS se skládá z několika vzájemně provázaných složek. Proces vyhledávání informací lze tedy chápat jako posloupnost činností jednotlivých složek, které fungují buď v on-line nebo off-line režimu. Off-line režim má v tomto smyslu význam práce se zpožděním nebo práce na pozadí, běží v něm procesy předzpracovávání a ukládání dat, naopak v on-line režimu probíhá jednak veškerá komunikace s tazatelem a dále zpracování dotazu, jeho porovnání s kolekcí dokumentů. Obr. 2.1 ilustruje zjednodušený průběh vyhledávání informací.



Obr. 2.1 – zjednodušený diagram DIS

2.1 Předzpracování dat

Předzpracování dat, neboli analýza obsahu, se zabývá rozborem a úpravou nasbíraných dat před uložením do databáze, lépe řečeno před jejich předáním indexační jednotce (součást systému, jež se stará o ukládání dat v co nejúspornější formě tak, aby se co nejvíce zrychlilo vyhledávání informací při minimální ztrátě sémantiky).

Kvůli zachování co největšího počtu významových jednotek v dokumentu řeší analýza obsahu otázky spojené s porozuměním textu. Problémy jsou s množstvím

nejednoznačností u vícejazyčných vyhledávacích strojů. Například v českém jazyce jde o synonymii slov (různá slova se stejným významem), homonymii slov (stejná slova s různými významy), časování, skloňování a podobně. Řešení takovýchto nejednoznačností spočívá ve využití metod lingvistické analýzy – disambiguice (určení správného významu slova ve větě na základě gramatiky nebo kontextu), lemmatizace (určení tvaru slova, slovního druhu, osoby, čísla atd. a využití větného rozboru) nebo hledání slovních spojení.

Druhou úlohou předzpracování je rozdělení dokumentu na entity – kusy textu, tokeny (běžně např. slova, věty, fráze) nebo rozdělení na významově odlišné sekce, což se ve výsledku může výrazně projevit při určování relevance vyhledávaných dokumentů.

2.2 Ukládání dat

Indexování předzpracovaných dat je klíčovou složkou vyhledávacích strojů, neboť právě volba vhodného indexovacího algoritmu se značnou měrou promítá na rychlosti vyhledávání. Porovnání dotazu s kolekcí dokumentů by nebylo časově efektivní. Mnohem lepších výsledků se dosáhne porovnáváním dotazu s dokumentem upraveným do jakéhosi modelu – speciální optimalizované datové struktury . To však s sebou přináší dvě nevýhody, které vznikají na úkor rychlosti vyhledávání. Jednak indexovaná data zabírají více místa v důsledku potřeby uložit s původním dokumentem i jeho model a jednak porovnávání probíhá pouze s modelem, jenž obvykle postrádá některé významové prvky původního textu, které algoritmus považuje za nevýznamné.

Indexační jednotka přijímá předzpracovaná data rozdělená do výše zmíněných entit. Postupy ukládání modelů, jinými slovy indexů, se různí vyhledávač od vyhledávače, v zásadě se ale v současnosti běžně používají následující metody:

- suffix tree – stromová struktura založená na ukládání přípon, nabízí lineární časovou závislost, nároky na uložení indexu můžou převyšovat nároky na uložení původních dat
- inverted index – datová struktura realizovaná nejčastěji pomocí takzvané transformační tabulky (hash table) nebo binárního stromu, obsahuje informace o dokumentech obsahujících dané slovo; existuje rozšířená varianta invertovaného indexu, která navíc nese informaci o pozici slova v dokumentu
- term dokument matrix – řídká dvourozměrná matice, kde jeden rozměr připadá

na dokumenty, druhý na slova v nich obsažená; používá se buď binární (pouze informuje o přítomnosti nebo nepřítomnosti slova v dokumentu) nebo vážená (zachycuje, kolikrát se slovo vyskytuje v dokumentu)

- n-gram – ukládá sekvence n entit (znaků, slov atd.), na základě těchto sekvencí lze s pomocí pravděpodobnostního rozdělení určit, jaká entita bude s největší pravděpodobností následovat; metoda n-gramů se dá použít takřka pro jakýkoliv typ dat

2.3 Vyhledávání a vizualizace výsledků

Při zadání vyhledávacího řetězce se tento nejprve upraví do podoby vhodné pro porovnání s indexovanými daty. Forma modelu dotazu souvisí jak s formou modelu dokumentů, tak s použitou metodou na získání kolekce relevantních dokumentů z databáze. V praxi se často používá latentní sémantické analýzy (latent semantic analysis – LSA) nebo relativně nového konceptu n-gramů. V případech rozsáhlých databank však může dojít k problému, že se podaří vybrat velké množství takzvaně relevantních dokumentů (takzvaně proto, že tazatelova představa o tom, co hledá, se může výrazně lišit od představy počítače, nehledě na to, že dokonce ani dva tazatelé se nemusí shodnout na důležitosti vybraného dokumentu) což může mít nepříznivý dopad nejen na tazatele, protože nebude ochoten procházet výsledky do hloubky, ale i na odezvu vyhledávacího stroje. Například webové vyhledávače uplatňují ještě další prvky pro zkvalitnění vyhledávání jako jsou výkladové slovníky, určování priorit podle pozice hledaného výrazu v textu (jestli se objeví v titulku, nadpisu, odstavci nebo citaci), různé varianty pageranků, tedy indexů (v tomto případě se indexem myslí bezrozměrné číslo) oblíbenosti stránek.

V neposlední řadě vstupuje do hry pojem shlukování, které je využitelné ke zkrácení času vyhledávání v bázi dat i pro jejich vizualizaci. Shlukování podobných nebo duplicitních dokumentů do skupin má výhodu v tom, že není nutné porovnávat dotaz s celou databází, stačí porovnat jej pouze se zástupci jednotlivých skupin. Stejně tak lze při zobrazování vypsat jen zástupce shluků a nechat tazatele rozhodnout, zda ho zajímají související dokumenty. Další výhodou je možnost automatizovaného rozdělení dokumentů do kategorií, mezi nimiž se snáze orientuje.

3 Shlukování

Obecně se v oblasti vyhledávání informací shlukováním (clustering) nazývá proces rozdělení skupiny dat do přirozených podskupin – shluků, kde členové takových skupin mají mezi sebou nějaký vztah, vyznačují se vzájemnou podobností. Navíc shlukovací algoritmus předem nezná uspořádání vstupních dat. Data mohou být reprezentována například body, respektive jejich souřadnicemi v rovině, prostoru apod., sekvencemi znaků, dokumenty nebo i složitějšími strukturami. Shluk, bez ohledu na druh dat, je tedy skupina vzájemně podobných objektů, avšak odlišných od objektů z jiných shluků.

Při rozhodování o podobnosti dvou prvků se v zásadě vychází ze dvou představ – první využívá matice vzájemných vzdáleností (distance-based) mezi stejnými druhy entit, matice má rozměr NxN; druhá představa (vector-based) definuje matici MxN, kde dimenze M odpovídá počtu prvků jednoho druhu entity a N počtu prvků druhého druhu. Distance-based konceptu podává informace o vzájemném vztahu mezi dvěma objekty, kdežto vector-based koncept informuje o vlastnostech každého jedince samostatně.

Hodnoty ve výše zmíněných maticích se získávají jako výstup z metrik. Existuje jich celá řada, nicméně pro density-based se stala populární Minkowského metrika:

$$d(x_r, x_s) = \{\sum_{k=1}^{p} \|x_{rk} - x_{sk}\|^q\}^{1/q}; \quad q \geq 1, \quad x_i \in \mathbb{R}^p \tag{3.1}$$

Vhodnou volbou parametrů z ní lze dostat například euklidovskou metriku (pro q=2). Ve většině případů má následující vlastnosti:

$$\begin{aligned} &d(x_r, x_s) \geq 0 \\ &d(x_r, x_r) = 0 \\ &d(x_r, x_s) = d(x_s, x_r) \end{aligned} \tag{3.2}$$

Vector-based koncept nachází využití ve shlukové analýze textových dokumentů v DIS, proto se hojně používá metoda vážení TF-IDF (term frequency – inverse document frequency) ve spojení s kosinovou metrikou. TF-IDF předpokládá, že čím častěji se výraz/slovo vyskytne v dokumentu, tím je důležitější. Frekvence výskytů (TF) jednotlivých slov t_i v dokumentech d_j se ještě normalizuje, aby se předešlo jevu, kdy se v kolekci vyskytne příliš dlouhý dokument, čímž by mohlo dojít ke zkreslení důležitosti slova vlivem vysoké frekvence výskytu. TF se tedy spočítá podle následujícího vzorce:

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{3.3}$$

kde $n_{i,j}$ odpovídá počtu výskytů *i-tého* slova v *j-tém* dokumentu a do čitatele se dosadí součet všech slov v tomtéž dokumentu. Ve druhé fázi se počítá IDF, neboli míra obecné důležitosti slova v kolekci dokumentů. Vzorec pro zjištění IDF následuje:

$$IDF_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|} \tag{3.4}$$

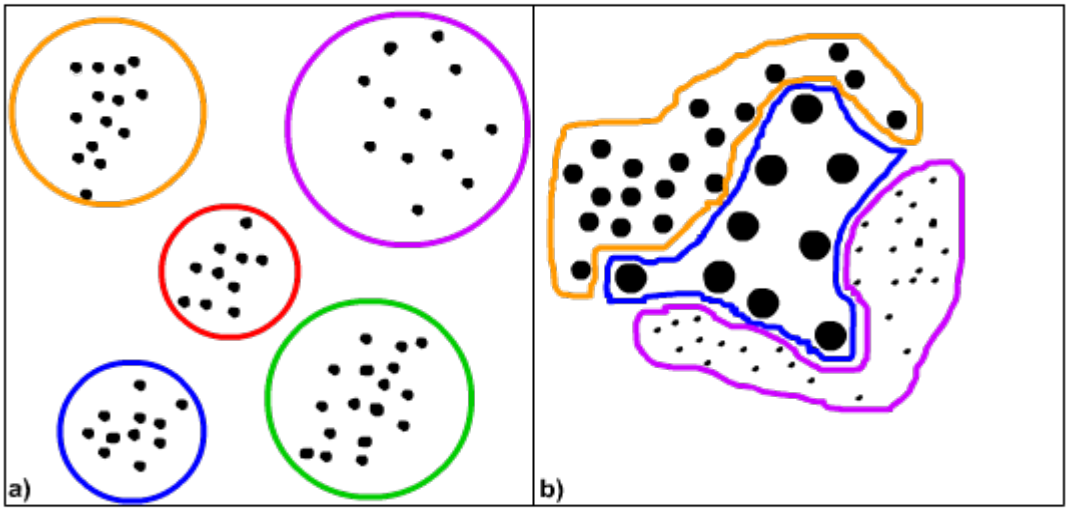
$|D|$ vyjadřuje celkový počet dokumentů ve zpracovávané kolekci, jmenovatel udává počet dokumentů, ve kterých se vyskytuje i-té slovo t_i. Nakonec se $TF_{i,j}$ a IDF_i mezi sebou vynásobí a dostaneme váhu důležitosti slova ve vybraném dokumentu. Vzájemná podobnost jednotlivých dokumentů se pak vyčíslí na základě kosinové metriky určující úhel mezi dvěma vektory *MxN* matice:

$$\theta = \arccos\left(\frac{A \,.\, B}{\|A\|\|B\|}\right) \tag{3.5}$$

Výsledný úhel podobnosti nabývá hodnot z intervalu $\langle 0 ; \pi \rangle$.

Při implementaci shlukovacích algoritmů se obvykle berou v potaz následující vlastnosti:

- podporované typy vstupních dat
- rozšiřitelnost na velké datové kolekce
- časová náročnost a náročnost na systémové prostředky
- závislost na pořadí dat
- závislost na vstupních parametrech zadávaných uživatelem
- schopnost přegenerovat data do nových shluků
- nároky na uložení shluků a mezivýpočtů
- překrývání shluků



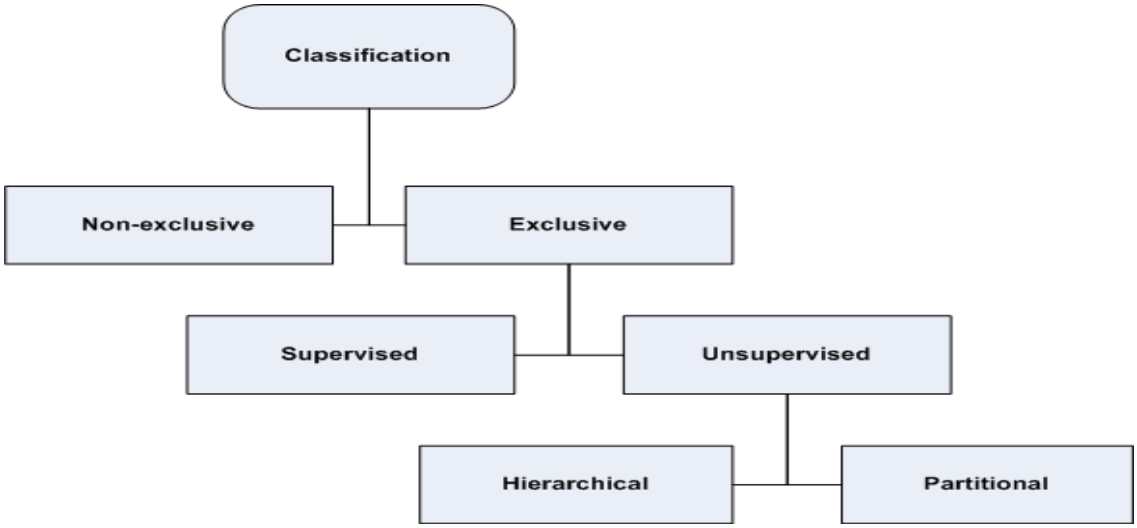
Obr. 3.1 – rozdělení bodů do shluků na základě a) vzájemné vzdálenosti, b) velikosti

3.1 Rozdělení shlukovacích algoritmů

Z hlediska shlukovacích algoritmů neexistuje jednoznačné stanovisko, podle kterého je dělit. Vždy záleží na úhlu pohledu. Chceme shlukování využít pro zpřehlednění výstupu vyhledávání nebo pro identifikaci míst se zvýšenou seismickou činností? Potřebujeme především kvalitní shluky a příliš nás netrápí časová náročnost, nebo potřebujeme vytvořit shluky v co nejkratším čase i za cenu zhoršení kvality? Pracujeme v on-line, nebo off-line režimu? Aplikujeme algoritmus na obsáhlou nebo spoře zaplněnou databázi? Způsobů rozdělení je skutečně mnoho. Základní linie rozdělení se však drží většina odborných publikací na téma shlukování ve souvislosti s vyhledáváním informací (viz obr. 3.2). Kromě graficky vyobrazených (dnes již tradičních) metod se tato kapitola v krátkosti zmiňuje o dalších způsobech shlukování – model-based, density-based a grid-based. Následují vysvětlivky k obr. 3.2:

- non-exclusive/overlapping – vytváří takzvané měkké shluky, tedy shluky, jenž se mohou vzájemně překrývat, to znamená, že jeden objekt se může vyskytovat ve více skupinách
- exclusive – oproti overlapping metodě vytváří tato tvrdé, uzavřené shluky, kde každý objekt patří pouze do jediného setu
- supervised – metoda vyžadující vstup uživatele v podobě například zadání počtu shluků, což může být nelehký úkol s často spornými výsledky
- unsupervised – tato kategorie shlukovacích metod nevyžaduje zásah uživatele

- hierarchical – data se shlukují postupně a stupňovitě
- partitional – vytváří shluky z kolekce dat naráz



Obr. 3.2 – stromová struktura základního rozdělení shlukovacích metod

3.1.1 Hierarchické algoritmy

Hierarchické shlukování tvoří hierarchii shluků nazývanou dendrogram, což napovídá, že jde o stromovou strukturu. Hierarchické metody se dále dělí na aglomerační (obr. 3.3.a) a na dělící (obr. 3.3.b). Aglomerační vychází ze stavu, kdy je každý atomický prvek datové kolekce členem svého vlastního shluku a postupně dochází ke slučování dvou nebo více nejvhodnějších shluků, dokud se algoritmus nedostane do stavu s jedním shlukem pro všechna data. V případě dělící metody se postupuje přesně opačně – tedy „od shora“ postupným dělením jednoho velkého shluku obsahujícího veškerá data až do doby, než počet potomků rodičovských shluků dosáhne nastavené hranice.

Většina algoritmů této skupiny využívá aglomeračního postupu při tvorbě shluků, jelikož dělící metody představují náročnou výpočetní úlohu a to především proto, že existuje $2^{n-1}-1$ možností, jak rozdělit data do dvou skupin. Z toho plyne, že počet možných kombinací rozdělení jde do velkých čísel i pro malé databáze.

Obecný algoritmus aglomerační metody vypadá následovně:

1. *Nechť každý objekt je považován zároveň i jako jednotkový shluk. Matice vzdáleností o rozměru NxN reprezentuje vzájemné vzdálenosti dvojic objektů.*

2. *Najdeme nejmenší prvek matice, jinými slovy vybereme dvojici vzájemně nejvíce podobných shluků h a i. Tyto shluky sloučíme*
3. *Spočítáme vzdálenosti mezi nově vytvořeným shlukem a těmi zbylými. Smažeme řádek a sloupec shluku h a přepíšeme řádek a sloupec shluku i novými hodnotami.*
4. *Pokud je počet shluků větší než stanovená hodnota k*[1] *, opakujeme postup od kroku 2, v opačném případě ukončíme činnost.*

Mezi výhody patří snadná implementace a rozumná rychlost výpočtu a benevolentní přístup k aplikovatelnosti různých metrik podobnosti, nicméně algoritmus má časovou náročnost $O(n^3)$. Další nevýhoda tkví v těsné závislosti právě na použité metrice podobnosti, což může znamenat produkci velmi odlišných shluků.

Obecný algoritmus dělící metody:

1. *Vyberme shluk obsahující dvojici nejvzdálenějších objektů, tedy shluk s největším průměrem (viz vzorec 3.6).*
2. *V tomto shluku vyberme objekt s největší průměrnou vzdáleností od ostatních objektů. Vyjměme tento objekt a umístěme ho do nového shluku.*
3. *Pro všechny objekty h ve shluku, který rozdělujeme, spočítejme průměrnou vzdálenost mezi objektem a tímto shlukem a průměrnou vzdálenost mezi objektem a novým shlukem. V případě, že vzdálenost mezi objektem a novým shlukem je menší než vzdálenost mezi objektem a původním shlukem, přesuňme objekt do nového shluku.*
4. *Pakliže jsme prošli všechny objekty, ale nedosáhli jsme požadovaného počtu shluků, pokračujeme znovu od bodu 1.*

$$D_{i,C_j} = n_j^{-1} \sum_{x_h \in C_j} d(x_i, x_h) \tag{3.6}$$

I tato metoda má své stinné stránky, přestože, nebo právě proto, že na základě doporučení Kaufmana a Rousseeuwa neprohledává ani zdaleka všechny možnosti rozdělení rodičovských shluků. Samozřejmě přepočítávání průměrů po každém přesunu objektu v bodě 3 vyžaduje nároky na počet operací a potřebnou paměť.

Vylepšení základních hierarchických metod nabízí například algoritmy BIRCH, CURE, CHAMELEON atd. Pro organizaci informací (kategorizaci), zejména pro použití ve webových vyhledávačích, byl vyvinut Star Clustering Algorithm.

1 Bývá vhodné nechat algoritmus pracovat až do vytvoření jediného společného shluku

3.1.2 Rozdělovací (partitional) algoritmy

Tradiční partitional algoritmy vyžadují určení *k* počtu shluků, mezi které se má kolekce dat rozdělit. Hlavní myšlenka spočívá v nalezení globálního optima účelové funkce. V tom případě je ale potřeba brát v potaz všechny způsoby rozdělení *n* objektů podle následujícího vzorce:

$$N(n,k)=\frac{1}{k!}\sum_{i=1}^{k}(-1)^{(k-i)}\binom{k}{i}i^{n} \tag{3.7}$$

Jenomže s rostoucím *n* se neúnosně zvětšuje počet možných kombinací jako v případě dělícího hierarchického algoritmu, což vede ke zjednodušení problému na hledání pouze lokálního optimálního řešení.

Mezi tradiční a dodnes oblíbené zástupce partitional algoritmů patří k-means popsaný již v roce 1975. k-means rozděluje objekty do *k* skupin tak, aby minimalizoval rozdíl mezi objektem a „těžištěm“. K měření rozdílů standardně používá normu L_2, čili se jedná o následující účelovou funkci:

$$A=\sum_{j=1}^{k}\sum_{x_i\in C_j}\|x_i-c_j\|^2 \tag{3.8}$$

kde x_i označuje míru *i-tého* objektu a C_j reprezentuje *j-tý* shluk s „těžištěm“ c_j.

Ukázka k-means algoritmu:

1. *Vytvořme k center („těžišť“) pro každý shluk.*
2. *Každý z n objektů přiřadíme do sluku, se kterým má nejmenší L_2 normu*
3. *Upravme centra shluků vhledem k současným členům*
4. *Pro každý objekt i, kde* $x_i\in C_j$ *spočteme* $h=arg\min_{r\neq j}\frac{n_r\|x_i-c_r\|}{n_r-1}$ *, kde* n_r *odpovídá počtu objektů přiřazených do shluku r.*

   Pokud $\frac{n_h\|x_i-c_h\|}{n_h-1}<\frac{n_j\|x_i-c_j\|}{n_j-1}$ *, pak přesuneme objekt i ze shluku j do shluku h. Upravme hodnoty relevantních center.*
5. *Ttt*

Obrovská výhoda k-means algoritmu spočívá v časové složitosti *O(n)*, umí pracovat s libivolnou L_p normou a je odolný vůči uspořádání dat. Na druhou stranu nutno podotknout, že se k-means snaží tvořit shluky stejných velikostí. Velmi také záleží na prvotním rozmístění center shluků, stejně tak na vhodně zvoleném počtu

požadovaných shluků[2]. K-means se příliš nehodí pro kategorizovaná data. To už neplatí pro k-medoids algoritmus. Od k-means se liší například tím, že centra shluků se náhodně vybírají přímo z množiny objektů, je šetrnější k paměťovým nárokům, stále ale vyžaduje určení počtu shluků a časová složitost je $O(n^2)$. K-means má i „měkkého“ kolegu, jinými slovy algoritmus tvořící shluky, jenž se mohou překrývat. Existuje několik vylepšení tradičních partitional metod, z nejznámějších třeba CLARANS (Clustering Large Applications based on a RANdomized Search), O metodách z rodiny cover-coefficient podrobně referuje kapitola Shlukování výsledků vyhledávání.

3.1.3 Model-based algoritmy

Tyto algoritmy se snaží formulovat model a naroubovat ho na kolekci dat tak, že se snaží odhadnout vhodné parametry. Většinou se mluví o statistických modelech nebo neuronových sítích.

Neuronové sítě využívají techniky samoorganizujících se map. Ty poprvé popsal Tuevo Kohonen, jenž se inspiroval přeskupováním neuronů v lidském mozku. Podle svého autora se neuronové sítě také nazývají Kohonenovy sítě nebo Kohonenovy mapy. Teorie vychází ze zajímavého problému: Neurony se v mozku rozšiřují v lineárním nebo dvourozměrném smyslu. Jenomže smyslové vnímání se odehrává ve vícerozměrném smyslu. Vstupní data se tedy mapují z vyšších dimenzí do maximálně dvourozměrné plochy. Metoda je to dvoufázová – v první fázi se algoritmus učí na vstupních příkladech, v druhé fázi automaticky mapuje vstupní data. Algoritmus může pracovat jak v inkrementálním tak dávkovém režimu. Jeden z konkrétních přístupů k algoritmu samoorganizujících se map se jmenuje WEBSOM.

Zástupcem kategorie metod využívajících statistických informací o datech je EM (Expectation Maximization). Algoritmus se opět skládá ze dvou kroků – E-Step a M- Step. EM představuje standardní přístup analýzy statistického model, který postrádá data. O datech se předpokládá, že jsou náhodným vzorkem ze směsi různých pravděpodobnostních rozdělení, jež definují shluky. Algoritmus trpí pomalým průběhem při větších objemech dat, jelikož musí při každém průchodu proskenovat celou kolekci dat.

2 Vhodný počet shluků lze zjistit podle výstupu hierarchické metody shlukování

3.1.4 Density-based algoritmy

Z názvu plyne, že density-based algoritmy se snaží vytvářet shluky definované jako hustě pokryté regiony v datovém prostoru. Tedy zhuštěné ostrůvky dat v jinak řídce rozprostřených datech.

DBSCAN – Density Based Spatial Clustering of Applications with Noise, velice populární algoritmus odolný proti přítomnosti šumu, s nízkou závislostí na pořadí dat, se složitostí $O(n.logn)$. Využívá myšlenky, že pro každý bod existuje určité okolí, ve kterém se sleduje hustota bodů. K vytvoření shluku je potřeba dosáhnout definovaného minimálního množství bodů v okolí vyšetřovaného bodu. Okolí se definuje pomocí poloměru.

Do této kategorie také zapadá algoritmus DENCLUE – DENsity-based CLUstEring.

3.1.5 Grid-based algoritmy

Jde o algoritmy rozdělující datový prostor do mřížky s konečným počtem buněk. Algoritmus STING (STatistical INformation Grid) pracuje se statistickými daty ukládanými do jednotlivých buněk. Vyžadovány jsou alespoň informace o počtu objektů v buňce a jejich střední hodnota. Dále lze ukládat směrodatnou odchylku, minimální a maximální hodnoty objektů a typ pravděpodobnostního rozdělení. Algoritmus shlukuje data na základě průchodů buněk a porovnávání hustoty buněk s dotazem. STING potřebuje celkem malý počet operací.

3.1.6 Ostatní algoritmy

Na úvod kapitoly 3.1 padla zmínka o neexistenci jednoznačného kritéria pro klasifikaci shlukovacích metod. Proto i tomuto krátkému přehledu uniká množství algoritmů označovaných za hybridní. Nezřídka kdy mezi ně patří ty, které se původně vešly do stanovených skupin, jenže kvůli nutné optimalizaci si museli vypůjčit myšlenky typické pro algoritmy z jiných kategorií.

3.2 Shlukování výsledků vyhledávání

Shlukování výsledků vyhledávání v textových informačních systémech se potýká se dvěma zásadními problémy. Za prvé není jednoduché procházet obrovským množstvím dat vrácených na dotaz, mezi kterými není nouze o duplikáty nebo tématicky velice blízké dokumenty, a za druhé každý uživatel může mít odlišnou

představu o relevantních výsledcích vyhledávání. Ukázka různých interpretací jednoho vyhledávacího dotazu „večer na horském plesu“ - buď tazatel hledá informace nadcházejícím o tanečním plesu konajícím se v horské chatě, nebo se snaží zjistit podrobnosti o nečekaném dobrodružství zabloudivších turistů při přenocování u horského jezera.

První případ se obvykle řeší nasazením výkonných algoritmů na hledání podobných dokumentů v kolekci. Takové algoritmy nejčastěji pracují v retrospektivním smyslu, jinými slovy v off-line režimu, ideálně při indexaci dokumentů. Vhodnými zástupci této kategorie se zdají být CC-based (Cover Coefficient) algoritmy – C^3M, C^2ICM a F^2ICM.

I na druhý problém existuje lék – kategorizace. Vhodným spojením dokumentů podobného zaměření do kategorií lze vytvořit přehlednou hierarchickou strukturu, ať už jako strom kategorií a podkategorií, nebo jako rovinnou mapu. Jde tedy především o shlukování pro potřeby vizualizace výsledků. Mezi často citované algoritmy se řadí WEBSOM a Star Clustering Algorithm. Těmi se však tato práce hlouběji nezabývá.

3.2.1 C^3M

C^3M (Cover Coefficient-based Clustering Methodology) znamená v překladu metoda založená na koeficientech pokrytí. Tento algoritmus je takzvaně seed-based. Slovo seed by se dalo volně interpretovat jako reprezentant nebo vedoucí. Celý proces probíhá ve třech fázích:

1. určení počtu shluků
2. výběr reprezentativních dokumentů (seed documents) z kolekce
3. Generování shluků

3.2.1.1 Určení počtu shluků

C^3M nevyžaduje volbu počtu shluků uživatelem. Algoritmus si vhodný počet dokáže zjistit sám. Samotnému výpočtu předchází několik dalších kroků, které se uplatní i v ostatních fázích.

Nejprve je potřeba vytvořit matici *dokument x výraz* (v literatuře nazývaná D Matrix) velikosti *m x n*, pro kterou platí, že řádky odpovídají dokumentům a sloupce jednotlivým unikátním výrazům ze všech dokumentů. Zároveň každý výraz musí být

obsažen alespoň v jednom dokumentu a každý dokument musí obsahovat alespoň jeden z výrazů. Pokud by tento předpoklad neplatil, vznikaly by při dalších výpočtech nedefinované výrazy (číslo dělené nulou). Takový případ může nastat, pokud se zapomene na nějaký dokument při parsování výrazů z dokumentů, nebo nedokonalým parsováním, jenž vede na paskvily výrazů. Pro každý dokument se do matice zanese počet výskytů každého jednotlivého výrazu v daném dokumentu. Praktikují se dva přístupy – podle jednoho, takzvaného binárního, se pouze určí, zda je výraz v dokumentu obsažen, tedy berou se hodnoty pouze z rozmezí *{0,1}*, druhý přístup uvádí počet všech výskytů výrazu v dokumentu. Prvky D matice se označují $d_{i,j}$, vektor výskytů slov v *i-tém* dokumentu d_i, a vektor výskytů *j-tého* slova v souboru dokumentů t_j.

Shlukovací metody založené na vektorovém prostoru určitým způsobem definují podobnosti mezi daty. U C^3M se počítají coupling koeficienty (ψ) a decoupling koeficienty (δ) a to pro dokumenty i pro výrazy. Decoupling koeficienty udávají míru, nakolik dokument není příbuzný v poměru k ostatním. Čím unikátnější dokument je (čím méně obsahuje výrazů společných se zbytkem souboru), tím je decoupling koeficient vyšší. Decoupling koeficient ještě potřebuje pro výpočet zjistit převrácené hodnoty součtů řádků (α) a převrácené hodnoty součtů sloupců (β) – viz 3.9 a 3.10.

$$\alpha_i = [\sum_{j=1}^{n} d_{i,j}]^{-1}; \quad 1 \le i \le m \tag{3.9}$$

$$\beta_j = [\sum_{k=1}^{m} d_{j,k}]^{-1}; \quad 1 \le k \le n \tag{3.10}$$

Nic už tedy nestojí v cestě výpočtu δ a ψ:

$$\delta_{D_i} = \alpha_i \sum_{j=1}^{n} (d_{i,j}^2 \beta_j) \tag{3.11}$$

$$\psi_{D_i} = 1 - \delta_{D_i} \tag{3.12}$$

Podobně se určí i coupling a decoupling koeficienty i pro výrazy:

$$\delta_{T_j} = \beta_j \sum_{i=1}^{m} (d_{i,j}^2 \alpha_i) \tag{3.13}$$

$$\psi_{T_j} = 1 - \delta_{T_j} \tag{3.14}$$

V posledním kroku se zjistí počet shluků (n_c) podle vzorce 3.15. Vypočtená hodnota skoro určitě nebude z oboru celých čísel, proto se musí zaokrouhlit. Počet shluků mimo jiné udává, kolik dokumentů se zvolí za reprezentanty.

$$n_c = \sum_{i=1}^{m} \delta_{D_i} \tag{3.15}$$

3.2.1.2 Výběr reprezentativních dokumentů

Za reprezentativní dokumenty budou vybrány ty, které zvítězí v „soutěži měření sil“. Nejprve se pro každý dokument definuje takzvaná síla reprezentanta[3] (seed power) podle 3.16 (pro váženou D matici) respektive 3.17 (pro binární D matici), následně se dokumenty seřadí podle síly sestupně a n_c dokumentů s největší silou se zvolí za reprezentanty. Pokud se po seřazení objeví skupina dokumentů se stejnou, nebo jen nepatrně rozdílnou hodnotou seed power, zvolí se za reprezentanta pouze jeden dokument z této skupiny.

$$P_i = \delta_{D_i} \psi_{D_i} \sum_{j=1}^{n} (d_{i,j} \delta_{T_j} \psi_{T_j}) \tag{3.16}$$

$$P_i = \delta_{D_i} \psi_{D_i} \sum_{j=1}^{n} d_{i,j} \tag{3.17}$$

3.2.1.3 Generování shluků

Ve třetí fázi postupu přichází na řadu proces přiřazování nereprezentativních dokumentů do vytvořených shluků. Za tímto účelem je nezbytné vytvořit matici koeficientů pokrytí (cover coefficient matrix), někdy označovanou jako C matrix nebo CC matrix. Tato matice má oproti D matici rozměr *m x m*, protože informuje o vzájemné podobnosti dvou dokumentů. Jednotlivé prvky matice se dostanou ze vzorce:

$$c_{i,j} = \alpha_i \sum_{k=1}^{n} (d_{i,k} \beta_k d_{j,k}) \tag{3.18}$$

Pro každý nereprezentativní dokument (řádek) poté porovnáme hodnoty ve sloupcích odpovídajících reprezentativním dokumentům a dokument přiřadíme do shluku, v jehož sloupci byla nalezena největší hodnota.

3.2.1.4 Použitelnost

C^{3}M algoritmus jako takový se nehodí pro inkrementální shlukování (případ kdy dokumenty přicházejí do systému postupně), nýbrž pro dávkové shlukování, jelikož by při příchodu nového dokumentu musel přegenerovat všechny shluky (více viz srovnání

3 Síla reprezentanta udává schopnost dokumentu přitahovat jiné do shluku, proto se vybírají jen dokumenty s největší sílou.

v praktické části). Naštěstí existují deriváty C^3M řešící inkrementální shlukování na základě již vytvořených shluků, na základě faktoru zapomínání nebo s pomocí časového okénka.

3.2.2 C3M s časovým oknem

Protože C3M je příliš „drahý“ pro nasazení například v systémech na shromažďování a prohledávání článků, objevilo se několik přístupů. Obvykle se tyto přístupy snaží vybrat jen nezbytně nutné množství již zaindexovaných dat. Ahmet Vural se nechal inspirovat C3M algoritmem a vytvořil derivát kontrolující stáří dokumentů přítomných v databázi a podle jejich stáří se rozhodne, jestli takové dokumenty vybere pro účely shlukování nových dokumentů.

Tato studie se zakládá na teorii, která tvrdí, že články referující o určité události, mají omezenou „trvanlivost“. Jinými slovy, po nějaké době přestávají být aktuální a vytěsňují je jiná témata. Toho lze s výhodou využít i pro shlukování dokumentů, jelikož stačí vybrat dokumenty z časového okna definované velikosti, čímž se velmi sníží nároky na pamět i výpočetní výkon. Ne všechny události se ale vejdou do časového okna a některé jej dokonce přesahují značně. Výsledkem je logicky snížení kvality shluků tím, že se může vyskytnout jeden nebo více shluků slučujících články o stejné události.

3.2.2.1 Algoritmus

Před samotným popisem algoritmu je třeba zmínit se o tom, jak odlišit reprezentativní dokument od nereprezentativního. U tradičního C3M se jednoduše spočítají síly jednotlivých dokumentů (viz 3.16 a 3.17), seřadí se sestupně podle velikosti sil a vybere se stejný počet dokumentů s největší silou, kolik algoritmus stanovil shluků. Jenže v případě verze s časovým okénkem nastává problém: starší dokumenty (tedy ty uložené v systému) jsou již rozděleny do shluků. Existuje možnost přegenerování shluků pro dokumenty v časovém okně vyjma těch, jejichž shluky přesahují časové okno, protože jinak by se z časového hlediska řetězovitě tvořily nežádoucí podlouhlé shluky. Ahmet šel ale jinou cestou a definuje jakousi hranici oddělující reprezentativní dokumenty od nereprezentativních. Podle něj nemá smysl mluvit o pevně definované hranici kvůli dopadům na kvalitu shluků, neboť dokumenty přicházející do systému se mohou značně lišit. Jako vhodnou metodu určení hranice (*Tr*)

zvolil Ahmet následující vzorec:

$$Tr = \sum_{d_i \in window} P_i / n_d \tag{3.19}$$

Kde P_i vyjadřuje sílu *i-tého* dokumentu (seed power) z časového okna a n_d počet dokumentů v časovém okně. Síla nově příchozího dokumentu se porovná s touto hranicí a pokud ji přesahuje, určí se nový dokument jako reprezentativní a přiřadí se do nového shluku.

V první fázi algoritmu se definuje časové okno odstraňují se všechny dokumenty, které se nevejdou do časového okna. V případě odstraňování reprezentativního dokumentu se odstraní i jeho shluk, nicméně ostatní dokumenty tohoto shluku, pokud se vejdou do časového okna, zůstávají.

V dalším kroku se spočítají seed power hodnoty pro jednotlivé dokumenty podle 3.16 nebo 3.17. Následně se určí hranice podle 3.19.

Pokud síla dokumentu přesahuje vypočtenou hranici, dokument se označí jako reprezentativní a zakládá nový shluk. V opačném případě, pokud již nějaký shluk existuje, spočítá se *i-tý* řádkový vektor matice koeficientů pokrytí (viz 3.18), tedy vektor odpovídající klasifikovanému dokumentu. Pakliže algoritmus nenalezne žádný již vytvořený shluk, umístí dokument do zvláštního shluku zvaného ragbag.

3.2.3 C²ICM

Další algoritmus odvozený od C³M se jmenuje Cover Coefficient-based Incremental Clustering Methodology. C³M s časovým oknem nechává jednou vytvořené shluky tak, jak jsou. Oproti tomu se C²ICM snaží urychlit práci právě při údržbě shluků, neboli při přeshlukování.

Přeshlukování celé kolekce dokumentů jako reakce na nově příchozí dokument by bylo příliš neefektivní. Navíc k tomu není důvod. Když se v systému objeví nový dokument, rozšíří se D matice o jeden řádek (dokument) a o žádný nebo několik sloupců (výrazů). Z předchozího souvětí tedy plyne, že i jediný dokument jistým způsobem zamíchá vztahy mezi jednotlivými dokumenty. Podle koncepce C²ICM pak už stačí jen vysledovat, které z dokumentů v kolekci byly ovlivněny natolik, že by se měly přeshlukovat. To se zajistí jednoduše. Stačí přepočítat síly dokumentů v kolekci včetně onoho nového dokumentu a zjistit, které z dokumentů se změnily z reprezentativních na nereprezentativní a naopak nebo pokud je reprezentant smazán

z kolekce.

C^2ICM má výhodu v tom, že si zachovává kvalitu shluků tvořených C^3M, jinými slovy že generuje shluky podobné těm generovaným C^3M. Navíc díky svým nízkým nákladům na přeshlukování může pracovat s obrovskými bázemi dat.

3.2.3.1 Algoritmus

První dvě fáze algoritmu, určení počtu shluků a výběr reprezentantů (seed dokumentů), se přesně shodují s postupy popsanými v kapitolách 3.2.1.1 a 3.2.1.2. Tím však podobnost končí a dále se postupuje podle kroků v dalším odstavci.

Nejprve se provede seřazení reprezentativních dokumentů získaných z předchozí fáze. Pak pokud se jedná o první shlukovací proces, vytvoří se shluky stejně jako v kapitole 3.2.1.3.

Pracuje-li algoritmus v režimu přeshlukování, potom se v první řadě vyberou reprezentanti z předešlého shlukování a porovnají se s reprezentanty aktuálními. Pakliže algoritmus zjistí, že původně reprezentativní dokument přestal být reprezentativním, označí se shluk, jehož je představitelem, za falešný (neplatný). Nyní se situace obrátí a v případě, že se mezi nově určenými reprezentanty nachází některý, který při předchozím (pře)shlukování nepatřil mezi reprezentanty, pak se shluk, jehož byl původně nereprezentativní dokument členem, označí za falešný.

Pro všechny zneplatněné shluky *y* se provede přeshlukování všech dokumentů *x* v *y*, pokud x existuje v databázi a patří mezi nereprezentanty. Pokud jeden nebo více reprezentantů sdílí stejné pokrytí nad *x*, přiřadí se *x* do shluku tvořeného reprezentantem (z těch, co mají stejné pokrytí nad *x*) s největší seed power. Pokud i zde dojde ke shodě sil seed dokumentů, přiřadí se *x* do shluku, jehož představitel má nejnižší číslo dokumentu.

3.2.4 F^2ICM

F2ICM (Forgetting Factor Incremental Clustering Methodology) koncept těží z myšlenek předchozích dvou přístupů. Ideu časového okna ale ještě více rozšiřuje, lépe řečeno přepracovává. Základním předpoklad zůstává stejný jako u C3M s časovým oknem, jenom je jinak vyjádřen. Jde opět o to, že čím je dokument starší, tím méně je aktuální, jinými slovy ztrácí váhu. Nové dokumenty mají vysokou váhu, zatímco

u starých se plynutím času rozmělňuje, jinými slovy staré dokumenty mají minimální vliv na shlukování. Čili se hovoří o faktoru zapomínání, se kterým se musí počítat při měření podobnosti mezi dokumenty.

Informační hodnota (dw_i) dokumentu d_i, tedy již zmíněná váha, exponenciálně klesá s časem a definuje jí následující vzorec:

$$dw_{i\tau} \equiv \lambda^{\tau - T_i} \tag{3.20}$$

kde T_i značí čas *i-tého* dokumentu, τ současný čas a λ definuje rychlost zapomínání a nabývá hodnot z otevřeného intervalu *(0 < λ < 1)*. Dokumenty se vybírají jen z určitého časového intervalu ε, respektive příliš staré dokumenty nezapadající do intervalu jsou odstraňovány z databáze. Proces generování shluků se odehrává v podobném duchu jako v případě C^3M s tím rozdílem, že při konstrukci C matice se každý prvek upravuje o normalizovanou váhu dokumentu podle 3.21. Dále není nutné při vkládání dokumentů do systému přepočítávat všechny hodnoty pro každý dokument znovu; u koeficientů vztahujících se ke konkrétnímu dokumentu stačí provést výpočty pouze pro nové dokumenty a u kombinovaných koeficientů existují způsoby „připočítání“ nových hodnot ke starým.

$$x \equiv \frac{dw_i}{\sum_{k=1}^{m} dw_k} \tag{3.21}$$

4 Testování vybraných algoritmů

Tato kapitola se zabývá porovnáváním konkrétních implementací vybraných algoritmů. Na úvod jsou rozebrány kapitoly přípravné fáze, které jsou společné pro všechny algoritmy. Prostřední část kapitoly se věnuje samotným implementacím jednotlivých algoritmů a na závěr proběhne jejich porovnání a zhodnocení.

4.1 Algoritmy

Rešerše na téma Shlukování výsledků vyhledávání vznikla ve spolupráci s Technickou univerzitou v Liberci a společností Seznam, a. s. a vzhledem k tomu, že Seznam hledá vhodné algoritmy k nasazení do svých služeb, jako třeba Články.cz – internetový portál shromažďující články ze zpravodajských serverů, padla volba na dvojici algoritmů nabízejících vysoký výkon – C^3M a inkrementální C^3M s časovým oknem.

4.2 Hardware

Všechny algoritmy byly testovány na notebooku následující konfigurace:

- procesor: Intel Pentium M, 1.73GHz
- operační paměť: 1024MB RAM
- pevný disk: 5400 ot./min., 74,5GB, volného místa 4GB

4.3 Software

Testování probíhalo výhradně na systému Microsoft Windows XP Professional SP2. Jako programovací jazyk pro implementaci samotných algoritmů byla zvolena Java, verze JRE 1.6.0_04 a pro vizualizaci shluků a prvotní analýzu testovacích dat se použil skriptovací jazyk PHP, instalovaný jako modul ve webovém serveru Apache. Java i PHP se těší velké popularitě mezi programátory, tudíž pro ně logicky existuje nespočet užitečných knihoven a kvalitní dokumentace.

Vývoj se odehrával v prostředí Eclipse, neboť je dostupné zdarma a nabízí veškerý komfort při programování v Javě. Vzhledem k jednoduchosti PHP skriptů stačil pro jejich zápis PSPad editor.

Pro každý algoritmus, potažmo parser dat, byla vygenerována samostatná spustitelná aplikace s příponou jar. Před každým testováním došlo k vynucenému

restartování počítače a po jeho opětovném spuštění se ukončily všechny procesy vyjma těch, co mají klíčový význam pro běh systému nebo těch, o kterých si autor nebyl jist jejich funkcí. Tím se zabránilo ovlivnění procesu testovací aplikace jiným příliš aktivním procesem.

Implementované algoritmy nepatří mezi rozsáhlé a přestože Java se řadí mezi striktně objektové jazyky, algoritmy působí spíše strukturalizovaným dojmem než objektovým.

Za datové úložiště se zvolily obyčejné textové soubory a MySQL databáze.

4.4 Testovací data

Testovací data poskytla společnost Seznam, a. s. Jedná se o výtah z databáze již zmíněné služby Články.cz, tedy o soubor dokumentů. Soubor obsahoval 10000 dokumentů a velkost činila 45,5MB. Dokumenty byly uloženy ve znakové sadě UTF-8. Každý řádek souboru odpovídal jednomu dokumentu s informací o datu a času příchodu do systému. Časová informace se nacházela na posledních devatenácti znacích každého řádku. Více než polovinu z celé velikosti zabíralo ohodnocení sekvencí znaků, podle toho, jestli šlo o posloupnost písmen nebo číslic, v takovém případě před každou posloupností stálo písmeno W (jako Word) a za ním dvouciferné šestnáctkové číslo udávající délku sekvence; nebo jestli šlo o posloupnost ostatních, nealfanumerických znaků, označených písmenem S (Space) a opět dvouciferným šestnáctkovým číslem.

4.5 Předzpracování dat

Data ze Seznamu se nedala použít pro shlukování okamžitě. Předzpracování dat se dělo za pomoci souborů cleanandfeed.php a clenaer.jar.

PHP skript, jak název napovídá, čistil surová data a plnil je do MySQL databáze. Nejprve odstranil hodnotící znaky Wxx a Sxx pomocí funkce split_and_clean(), jež očekává vstupní parametr v podobě jednoho řádku souboru, protože jejich odstraňování při vizualizaci shluků by se časově nevyplatilo. Následně čisté články uložil do databáze (tabulka *documents*) včetně diakritiky, interpunkčních znamének a ostatních znaků označených jako mezera.

Pro potřeby konstrukce D matice PHP skript dále rozdělil funkcí get_words_only() vstupní data na posloupnost slov oddělených mezerou a ukládal je opět do databáze (tabulka *preprocessed_stop*), ovšem do jiné tabulky. Úspora místa

v porovnání s 24,9MB u čistých článků činila 0,8MB. Funkce požaduje přinejmenším jeden vstupní parametr a sice jeden řádek surových dat. Druhý, volitelný parametr očekává pole takzvaných stop words, neboli nevýznamových slov (každý prvek pole musí být ve tvaru pro použití jako regulární výraz kompatibilní Perlem). To jsou slova často se objevující v textu a tudíž mají význam šumu, který zkresluje skutečnou informaci o klasifikovaném dokumentu. Tady už je úspora místa patrná – 3,1MB oproti čistým článkům, navíc tabulka s články neobsahuje informaci o datu, kdežto tabulka s články v podobě sekvence slov ochuzených o nevýznamová slova ano.

D matice se definuje jako matice s počtem dokumentů v jedné dimenzi a počtem unikátních výrazů napříč dokumenty v dimenzi druhé. Taková matice ale může u větších databází narůst do obrovských rozměrů, proto se praktikuje metoda na omezení počtu unikátních výrazů. Samozřejmě tak, aby nedošlo ke zbytečným ztrátám informací. Metoda se nazývá stemming a cílem je normalizace slov, tedy odstraňování pádů, skloňování, časování, přípon, koncovek, které se liší v každém rodu atd. Jinými slovy se stemmer, jak se programu provádějící stemming říká, snaží o úpravu slov tak, aby se výsledek co nejvíce blížil kořenu daného slova, nebo se alespoň upravené slovo nelišilo v různých případech použití ve větě. I pro účely této práce byl použit stemmer, konkrétně implementace algoritmu představeném v projektu normalizace slov vybraných východoevropských jazyků. Implementace již neproběhla v PHP, nýbrž v Javě (cleaner.jar).

4.6 Implementace algoritmů

Vývoj v jazyce Java probíhal po nastudování potřebných materiálů a každý algoritmus se může pochlubit několika verzemi, z nichž představeny budou jen poslední funkční verze. Všechny tři postupy používají D matici, aby vyjádřily vztahy mezi dokumenty a výrazy. Nicméně D matice má obecně charakter řídké matice, což vedlo autora k použití balíčku pro manipulaci s maticemi, v němž se mimo jiné nacházejí třídy pro konstrukci a práci s řídkými maticemi, které šetří místo nutné pro uložení matice do paměti, jelikož alokují místo pro prvek až ve chvíli jeho vytvoření. V Javě sice existují vícerozměrná pole, jenže se jedná o falešná vícerozměrná pole. Datový typ Array totiž v Javě nepatří mezi primitivní datové typy, to znamená, že pokud se definuje dvourozměrné pole, ve skutečnosti se definuje pole polí. A to si s sebou nese jistá úskalí:

1. při inicializaci například „dvourozměrného“ pole pro uložení D matice se alokuje paměť pro všechny prvky pole
2. nejrychlejších výpočtů se dosáhne tehdy, kdy jsou všechny prvky pole v paměti umístěny vedle sebe, což v případě pole objektů nemusí platit a obvykle také ani neplatí

Existuje několik konceptů definujících struktury pro ukládání řídkých matic, základní představa ukládá řídké matice do spojových seznamů. Nicméně není třeba znovu vymýšlet kolo, takže se použilo již hotové řešení ze zmíněného balíčku tříd.

Všechny naprogramované algoritmy se vyznačují totožnou strukturou čítající trojici tříd:

- spouštěcí třída (Main nebo RunClass) – vytváří instanci algoritmu a spouští klíčové metody
- třída s algoritmem (C3M4 nebo C3Mw) – definuje veškeré metody konkrétních algoritmů
- třída pro zapisování důležitých informací o běhu programu do logovacího souboru (Logger)

4.6.1 C^3M

Následující body popisují postup práce implementovaného algoritmu C^3M. Poslední verze umí pracovat jak s váženou D maticí, tak i s binární. Na úvod je ještě potřeba podotknout, že rozdíl mezi seznamem (Link) a množinou (Set) tkví v tom, že množina nepovoluje duplicitní prvky.

4.6.1.1 Třída C3M4

- metoda LoadResults():

  - tady se odehrává přípravná fáze před samotným shlukováním; nejprve se program spojí s MySQL databází pomocí konektoru z oficiálního balíčku vydávaného výrobcem MySQL databáze.

  - dále se vybere požadovaný počet dokumentů z databáze, které se uloží do pole řetězců

- metoda feedSet():

  - zde se prochází pole řetězců, jednotlivé řetězce se rozdělují na slova

a ta se následně vkládají do pomocného pole a do množiny slov

- nakonec metoda nastavuje pole řetězců na NULL, což dává signál garbage collectoru k uvolnění paměti alokované tímto polem

- constructDMatrix():

  - v první řadě se zjišťuje počet dokumentů ve výběru (nelze použít hodnotu předanou metodě LoadResults(), jelikož databáze může vrátit menší počet dokumentů) a množství slov v množině

  - v dalším kroku program prochází množinu slov a každé slovo porovnává se slovy z pomocného pole; pro binární matici platí, že při prvním výskytu slova z množiny v pomocném poli se ukončí porovnávací smyčka a program pokračuje dalším slovem

  - kvůli šetření prostředků se při procházení množiny zároveň počítají řádkové a sloupcové součty

  - nakonec se vynuluje pomocné pole

- constructCCMatrix():

  - na začátku se spočtou převrácené hodnoty řádkových a sloupcových součtů a inicializují se pole pro coupling a decoupling koeficienty dokumentů i výrazů

  - v rámci výpočtu koeficientů pro dokumenty se jako vedlejší produkt zjišťuje počet shluků k vytvoření, následně se spočítají koeficienty i pro výrazy

  - následuje výpočet síly dokumentu (pro binární i váženou D matici) a seřazení sestupně podle velikosti

  - před samotným výpočtem C matice (v programu označena jako Ccmatrix) se ještě volá privátní metoda determineSeeds(), která nedělá nic jiného, než že zjišťuje, jestli se mezi reprezentativními dokumenty nenacházejí dva nebo více se stejnou silou, pokud takovou *n-tici* metoda nalezne, vybere z ní pouze jednoho reprezentanta

  - výpočet matice je jednoduchý, navíc není nutné počítat C matici celou, neboť se pouze porovnávají nereprezentativní dokumenty s reprezentanty, čili stačí vypočítat submatici o rozměrech $(m - n_c)x(n_c)$.

- CreateClusters():

  - poslední klíčová metoda provádějící porovnání nereprezentantů s reprezentanty, na jehož základě provádí rozdělení do shluků

- třída C3M4 dále obsahuje skupinu metod naprogramovaných jen a pouze pro ladění aplikace, tudíž nemá smysl se o nich blíže zmiňovat

4.6.1.2 Třída Main

- vytváří novou instanci objektu Logger a zároveň novou instanci C3M4 objektu
- postupně volá všechny zásadní metody popsané v kapitole 4.6.1.2 a měří dobu provádění celého shlukování v nanosekundách
- průběžně informuje o dokončených činnostech
- na závěr nuluje instanci C3M4, aby mohl být uvolněn z paměti

4.6.1.3 Spuštění aplikace a komunikace s uživatelem

Aplikace je exportovaná do JAR balíčku, který obsahuje jednak potřebné externí knihovny (MySQL connector a balíček pro práci s maticemi) a jednak balíček v vlastním algoritmem.

JAR balíček se spouští z příkazové řádky coby jeden z parametrů aplikace java.exe. Spouštěcí metoda Main algoritmu vyžaduje tři vstupní parametry zadané jako argumenty jar balíčku. První určuje, zda se bude počítat s binární nebo váženou maticí a může nabývat hodnot *true* nebo *false*. Druhý parametr (typu String) udává název logovacího souboru a poslední parametr (typ Integer) říká, na kolika dokumentech se má provést shlukování. Není také na škodu předat Java Virtual Machine informaci o předpokládané velikosti takzvané Heap Memory (paměti vyhrazené pro běh aplikace). Příklad spuštění JAR balíčku s algoritmem následuje ilustruje obrázek 4.1.

Obr 4.1 – ukázka spuštění implementovaného algoritmu s 512MB heap memory, binární reprezentací D matice, logovacím souborem „log.log“ a 1000 dokumenty ke shlukování

Aplikace nemá grafické uživatelské rozhraní a to z důvodu vyšší rychlosti počítání a z důvodu, že byla vytvořena pro účely testování a o vizualizaci generovaných shluků se stará PHP skript generující HTML stránku se shluknutými dokumenty.

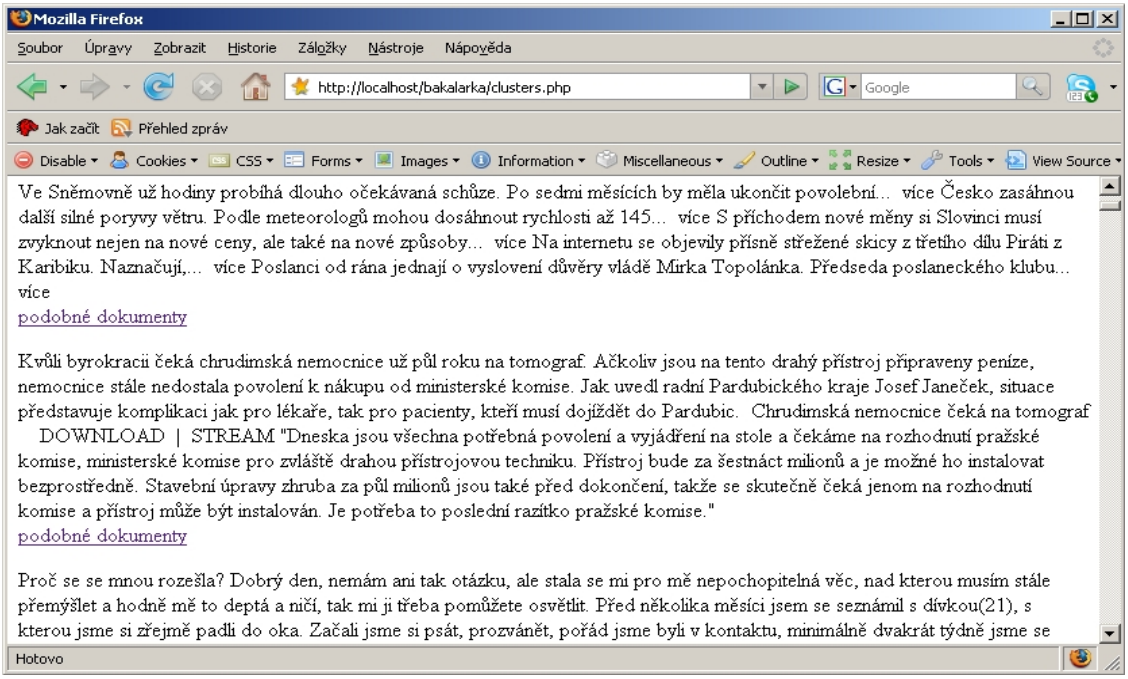
4.6.1.4 Průběh testování

Testování aplikace mělo tři fáze:

1. Zahřívací kolo – spuštění aplikace s malým počtem dokumentů (100 dokumentů)
2. Ostrý výpočet – aplikace byla 10x spuštěna, pokaždé s jiným počtem dokumentů (100, 200, 300, …, 1000), použita byly obě varianty konstrukce D matice pro porovnání rychlosti konstrukce a kvality shluků
3. Testovací shlukování – shlukování provedeno s padesátkou dokumentů pro účely vyhodnocení kvality shluků

4.6.1.5 Vizualizace shluků

Zobrazení shluků bylo provedeno pomocí PHP skriptu (clusters.php a similar.php). Ten načítá soubor, kde jeden řádek odpovídá jednomu shluku. Každý řádek začíná názvem shluku (např. *Cluster0*), následuje dvojtečka, mezera a dál seznam dokumentů patřících do shluku, kde vždy první z nich je reprezentant. Dokument se značí *D0, D1 ... DN*, kde *N* je číslo dokumentu. Vygenerovaná HTML stránka zobrazuje seznam reprezentativních článků a u těch článků, v jejichž shlucích se nachází alespoň jeden nereprezentant, se vypisuje odkaz na stránku s podobnými dokumenty. Výstup z vizualizátoru ukazuje následující obrázek:



Obr. 4.2 – zobrazení seznamu reprezentativních článků z vygenerovaných shluků

4.6.2 C^{3}M s časovým oknem

Na první pohled se kód zdá velmi podobný kódu předchozího algoritmu. Aby taky ne, když oba dva vycházejí z podobného základu. Tato implementace se však malinko liší od originálního algoritmu Ahmeta Vurala. Za prvé se nemažou staré dokumenty, protože ne vždy je žádoucí mazat archívní data, za druhé – jako důsledek první odlišnosti – se do časového okna nezahrnou ty dokumenty, jejichž představitel (seed dokument) se do časového okna také nevešel. Pokud by v časovém okně takové dokumenty zůstaly, mohly by způsobit řetězení událostí a zpochybňovaly by původně vytvořené shluky, navíc reprezentativní dokument by mohl snadno ztratit svoji výsadu nejsilnějšího dokumentu.

V případě C3M byla výpočetní logika programu oddělená od přípravy dat pro D matici. U C3M s časovým oknem ale příprava dat hraje velkou roli a prolíná se s hlavní logikou programu.

4.6.2.1 Třída C3Mw

- oproti třídě C3M, tato ukládá vygenerované shluky do databáze, namísto do textového souboru, za tímto účelem využívá tabulky *c3mw_clusters*
- třída dále využívá tabulky *c3mw*, aby mohla simulovat příchod nových dokumentů do systému
- LoadResults():

  - důležitým vstupním parametrem je *index* – ukazatel toho, kolikátý článek se má vybrat z tabulky předzpracovaných dokumentů; vybraný dokument se pak bude chovat jako nově příchozí do systému

  - nejprve se zjistí počet shluků uložených v databázi a podle nich se určí číslo potenciálního nového shluku

  - vybere se „nově příchozí“ dokument, zjistí se o něm informace (číslo dokumentu, čas příchodu) a umístí se jako první dokument do množiny dokumentů

  - vyberou se reprezentativní dokumenty, které se vejdou do hodnoty časového okna předaného konstruktoru, k nim se vyberou ostatní dokumenty patřící do shluku a uloží se včetně reprezentantů do množiny dokumentů, zaznamená se také informace o členství ve shlucích

- na závěr metody se množiny dokumentů a shluků převedou na pole pro snazší přístup k jednotlivým prvkům

- feedSet():
  - viz 4.6.1.1
- constructDMatrix():
  - opět stejné jako v 4.6.1.1
- constructCCMatrix():
  - až po výpočet síly dokumentů totožná s C3M
  - podle definice 3.19 se na základě podílu součtu sil dokumentů z časového okna a jejich počtu vypočítá hranice
  - pokud síla nového dokumentu převyšuje hranici, pak to znamená, že byl nalezen představitel nového shluku a stačí jen vytvořit nový shluk v databázi a uložit dokument mezi ostatní
  - pokud je síla nižší než stanovená hranice, nezbývá než vypočítat vektor C matice pro tento nový dokument; dokument se posléze zařadí mezi ostatní a podle toho, se kterým dokumentem má nejlepší koeficient pokrytí, se vybere shluk, do něhož se přiřadí.

4.6.2.2 Třída RunClass

Třída RunClass funguje podobně jako třída Main (viz 4.6.2.1) s tím rozdílem, že hlavní kód algoritmu je v RunClass vykonáván uvnitř cyklu, který má na starost inkrementování indexu a tím postupné procházení kolekce předzpracovaných dokumentů.

4.6.2.3 Spuštění aplikace a komunikace s uživatelem

Implementace C^3M s časovým oknem opět nedisponuje grafickým uživatelským rozhraním. Aplikace se spouští opět z příkazové řádky a vyžaduje předání čtyř argumentů – po řadě: přepínač mezi binární/váženou maticí (hodnoty *true/false*), název souboru pro ukládání logů, počet dokumentů ke shlukování a velikost časového okna v sekundách.

Zobrazení se řeší PHP skripty *clusters2.php* a *similar2.php*. Výsledkem je HTML stránka mající společné parametry s tou, generovanou skripty *clusters.php* a *similar.php*

4.6.2.4 Průběh testování

Testování probíhalo ve třech fázích:

1. Zahřívací kolo s padesáti dokumenty, ze kterého vzešly shluky pro účely testování kvality
2. Ostrý výpočet pro 250 dokumentů - výpočet s váženou maticí; experiment byl proveden s časovým oknem 10 dní a byl omezen pouze na 250 dokumentů, protože od 140 dokumentu měly všechny následující dokumenty stejnou časovou známku, tudíž časové okno nemělo dále na shlukování vliv

4.6.2.5 Vizualizace

PHP skript získává informace o generovaných shlucích z MySQL databáze a zobrazuje je stejným způsobem jako v případě implementace C^3M.

5 Výsledky testování

Testování bylo provedeno na dvou frontách. První fronta studovala časovou a paměťovou náročnost shlukování různého počtu dokumentů, druhá se zabývala počtem a kvalitou shluků.

5.1 Srovnání časové a paměťové náročnosti

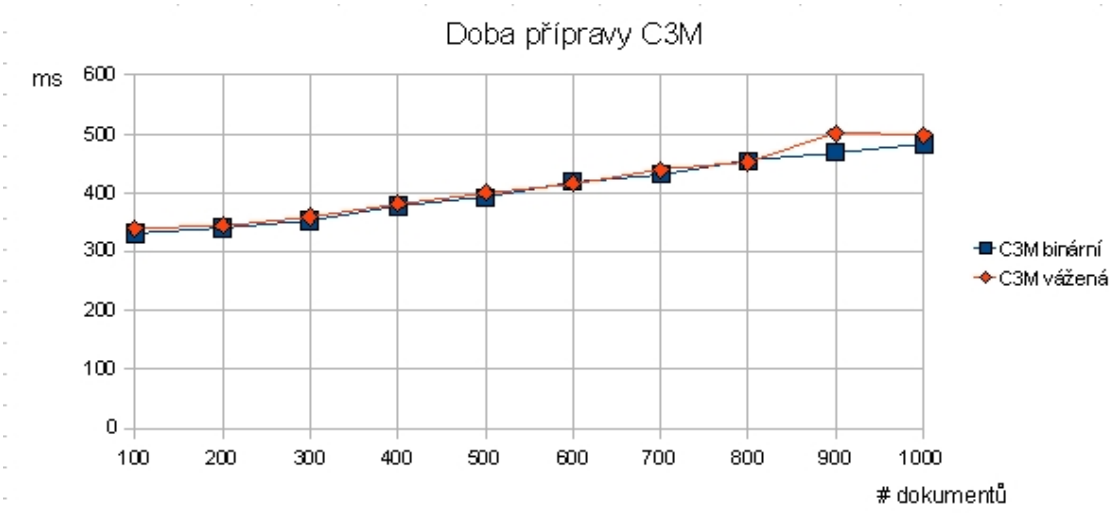
U C^{3}M se mění počet dokumentů po velkých hodnotách vzhledem k návrhu algoritmu, který by musel při každém novém dokumentu přeshlukovat celou kolekci. Místo toho se časová a paměťová náročnost měřila v deseti okamžicích a algoritmus tedy pracoval v režimu dávkovém.

Naproti tomu je C^{3}M s časovým oknem navržen pro inkrementální shlukování, tudíž nebyl důvod měřit náročnost po skocích, nýbrž po jednom dokumentu. Aby byly hodnoty i tabulkově porovnatelné, počítaly se kumulativní součty vždy po klasifikaci 100 dokumentů.

Dále je přiložen graf reálných nákladů u shlukování inkrementálním algoritmem.

Tabulka 5.1 – srovnání doby přípravy dat pro vytvoření D matice

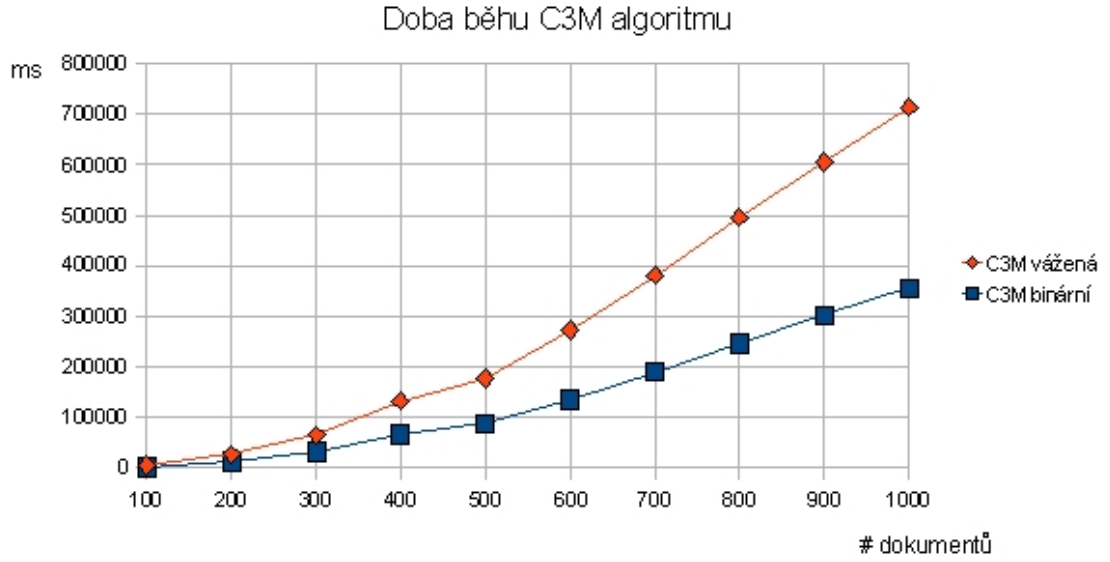
	Doba přípravy v milisekundách									
	100	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
C3M binární	331	340	353	377	392	419	431	454	469	483
C3M vážená	339	344	359	382	400	416	438	452	501	497
C3M s oknem	11306	22958								



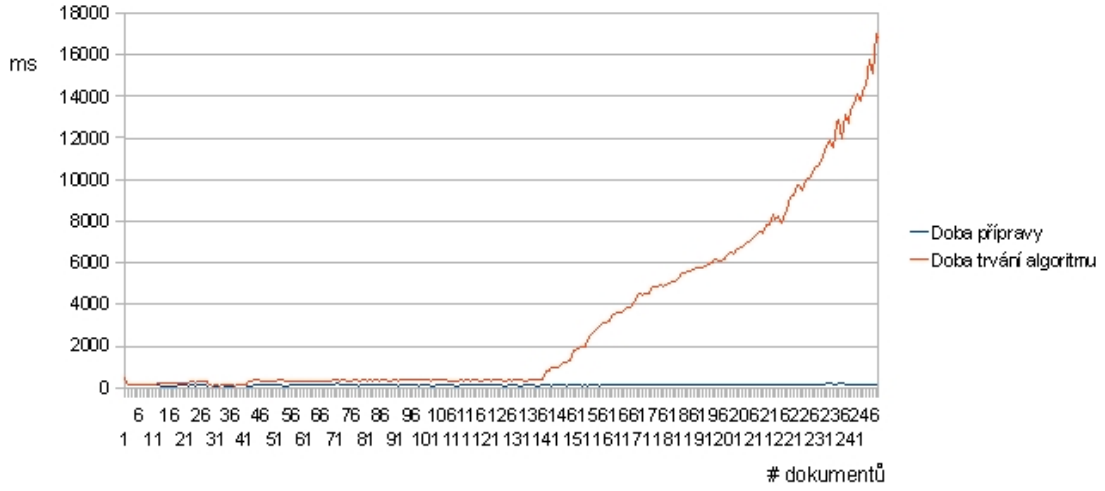
Graf 5.1 – znázornění doby přípravy dat pro algoritmus C3M

Tabulka 5.2 – srovnání doby trvání algoritmů

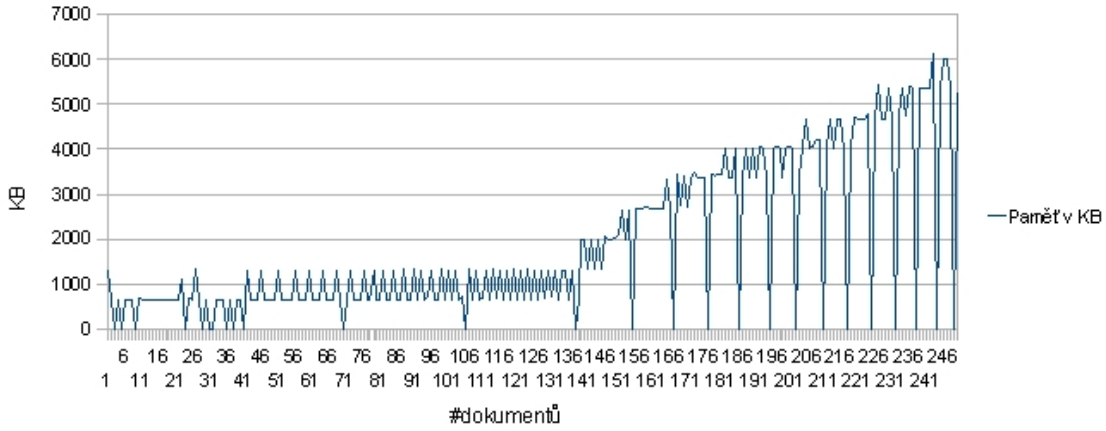
	Doba trvání algoritmu v milisekundách									
	100	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
C3M binární	2393	13063	32510	66096	88695	136090	189892	245853	301685	356050
C3M vážená	2514	13713	32417	66692	89233	137088	191225	249620	303143	357259
C3M s oknem	28021	279092								



Graf 5.2 – křivky vyjadřující dobu běhu algoritmu C3M



Graf 5.3 – časová náročnost inkrementálního C³M algoritmu s časovým oknem s patrným bodem zlomu kolem 140. dokumentu



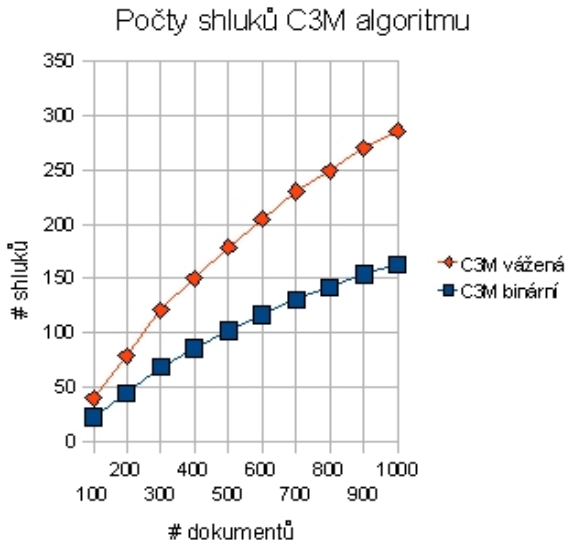
Graf 5.4 – využitá paměť při shlukování metodou s časovým oknem

5.2 Určení počtu a kvality shluků

Vyhodnotit počet shluků vygenerovaných v kontrolních bodech je to nejmenší, jde o čistě aritmetické počty. Na druhou stranu ale jak říct, jestli je shluk kvalitní nebo ne. Neexistuje žádná definice dobrého shluku. To závisí čistě na subjektivním dojmu uživatele. Při testování implementace C³M algoritmu vyšlo najevo, že do prvního sluku se zařazuje většina nově příchozích dokumentů, zatímco ostatní shluky mají nízkou „přitažlivou sílu“. To může být způsobeno buď chybou programu, nebo přílišnou benevolencí při přiřazování dokumentů do shluků. Řešením by mohlo být zavedení určité hranice založené například na seed power, podobně jako u C³M s časovým oknem.

	Počty shluků									
	100	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
C3M binární	23	45	69	86	102	117	131	142	154	163
C3M vážená	17	34	52	64	77	88	99	107	116	123
C3M s oknem	20	61								

Tabulka 5.3 – počty vygenerovaných shluků testovanými algoritmy



Graf 5.5 – počty shluků C³M algoritmu

5.3 Zhodnocení měření

Z měření vyplývá několik zajímavých informací. Doba přípravy, tedy čas, kdy se data získaná z databáze nahrávají do množiny dokumentů, roste u algoritmu C³M jen velmi mírně, u algoritmu s časovým oknem takřka vůbec (výjimkou je pouze výběr prvního dokumentu, který je ale možno vzhledem k deviačním vlastnostem vynechat), čas přípravy kmitá kolem 100 milisekund. Z toho plyne, že práce s databází má jen minimální vliv na dobu počítání algoritmů.

Na rozdíl od doby přípravy, křivka celkové doby trvání shlukování připomíná exponenciálu. To jen dokazuje fakt, že originální C³M algoritmus nemá ideální vlastnosti pro inkrementální shlukovací algoritmus. Nicméně naměřené časy se určitě dají snížit vhodnou optimalizací kódu, použitím low-level programovacího jazyka nebo nasazením na stroji s podporou rychlých vektorových operací. Experiment C³M algoritmu s časovým oknem sice vykazuje konstantní relativně nízkou dobu výpočtu, nicméně znehodnocením experimentu se ukázalo, že vyskytne-li se v souboru větší

počet dokumentů s podobnou časovou známkou, časová náročnost se začne rapidně zvyšovat, podobné chování lze vypozorovat i z křivky závislosti využité paměti na počtu dokumentů (viz Graf 5.3 a 5.4)

Sledováním vývoje křivky počtu shluků C3M algoritmu lze dojít k závěru, že s přibývajícími daty bude klesat míra nárůstu počtu shluků a pomalu se bude blížit určité hodnotě.

6 Závěr

Za pomocí experimentů na dvou implementacích shlukovacích algoritmů se podařilo vysledovat, jakého trendu by se tyto algoritmy držely při nasazení ve větších kolekcích dat. A co víc, výsledky ukázaly, že největší síla je v algoritmech založených na faktoru času, protože například myšlenka faktoru zapomínání vznikla inspirováním se nejen v reálném světě, ale i ve světě neurologie, neboť schopnost mozku zapomínat závisí na čase. Na druhou stranu experiment ukázal na slabinu, která by sice nastala výjimečně, avšak s o to horšími následky.

Seznam použité literatury

[1] BERRY, Michal W. – CASTELLANOS, Malu. *Survey of Text Mining II – Clustering, Classification, and Retrieval.* London: Springer-Verlag, c2008. ISBN 978-1-84800-045-2

[2] BORGATTI, Stephen P. *How to Explain Hierarchical Clustering* [online]. c1994. <http://www.analytictech.com/networks/hiclus.htm>

[3] *DBSCAN – Wikipedia, the free encyclopedia* [online]. 16. 10. 2007, poslední revize 27. 3. 2008. <http://en.wikipedia.org/wiki/DBSCAN>

[4] DOLAMIC Ljiljana – SAVOY, Jacques. *Stemming Approaches for East European Languages*. 2007. 12 s. Switzerland: Uviversity of Neuchatel. Computer Science Department

[5] HEIMSUND, Bjørn-Ove. *Matrix Toolkits for Java (MTJ)* [online]. <http://ressim.berlios.de/>

[6] KOGAN, Jacob – NICHOLAS, Charles – TEBOULLE, Marc. *Grouping multidimensional data – Recent Advances in Clustering.* Berlin: Springer-Verlag, c2006. ISBN 978-3-540-28348-5

[7] KOHONEN, Teuvo. *SOM Toolbox: Intro to SOM by Teuvo Kohonen* [online]. 18. 3. 2005. <http://www.cis.hut.fi/projects/somtoolbox/theory/somalgorithm.shtml>

[8] KOPECKÝ, Michal. *Dokumentografické informační systémy* [online]. 31. 3. 2008 <http://www.ms.mff.cuni.cz/~kopecky/vyuka/dis/html/siframes.html>

[9] *Information retrieval – Wikipedia, the free encyclopedia* [online]. 22. 11. 2001, poslední revize 13. 5. 2008. <http://en.wikipedia.org/wiki/Recall_ %28information_retrieval%29>

[10] MERCER, D. P. *Clustering large datasets*. 2003. 50 s. Linacre College.

[11] STREHL, Alexander. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining* [online]. 3. 5. 2002. <http://www.lans.ece.utexas.edu/~strehl/diss/htdi.html>

[12] VURAL, Ahmet. *On-line new event detection and clustering using the concepts of the cover coefficient-based clustering methodology*. 2002. 81 s. Bilkent university.

Institute of engineering and science. Department of computer engineering. Advisor Prof. Dr. Fazlı Can.

[13] ZHU, Xinpei. *Automatic Hypertext Link Creation Based on Clustered Nodes*. 2003. 99 s., 70 s. příloh. Oxford, Ohio: Miami University. Department of Computer Science and System Analysis. Advisor Dr. Fazli Can

[14] *Canopy clustering algorithm – Wikipedia, the free encyclopedia* [online]. 2. 12. 2007, poslední revize 24. 12. 2008. <http://en.wikipedia.org/wiki/Canopy_clustering_algorithm>

[15] *Cluster analysis – Wikipedia, the free encyclopedia* [online]. 21. 5. 2004, poslední revize 4. 5. 2008. <http://en.wikipedia.org/wiki/Data_clustering>

[16] *Clustering – Introduction* [online]. 17. 9. 2003. <http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/index.html>

[17] *Expectation-maximization algorithm – Wikipedia, the free encyclopedia* [online]. 15. 2. 2004, poslední revize 2. 5. 2008. <http://en.wikipedia.org/wiki/Expectation-maximization_algorithm>

[18] *Internet Usage World Stats – Internet and Population Statistics* [online]. *c*2008. <http://internetworldstats.com/>

[19] *K-means algorithm – Wikipedia, the free encyclopedia* [online]. 9. 5. 2005, poslední revize 5. 5. 2008. <http://en.wikipedia.org/wiki/K-means>

[20] *Self-organizing map – Wikipedia, the free encyclopedia* [online]. 2. 12. 2007, poslední revize 24. 12. 2008. <http://en.wikipedia.org/wiki/Self-organizing_map>

[21] *Single lincage clustering – Wikipedia, the free encyclopedia* [online]. 31. 8. 2007, poslední revize 11. 2. 2008. <http://en.wikipedia.org/wiki/Single_linkage_clustering>